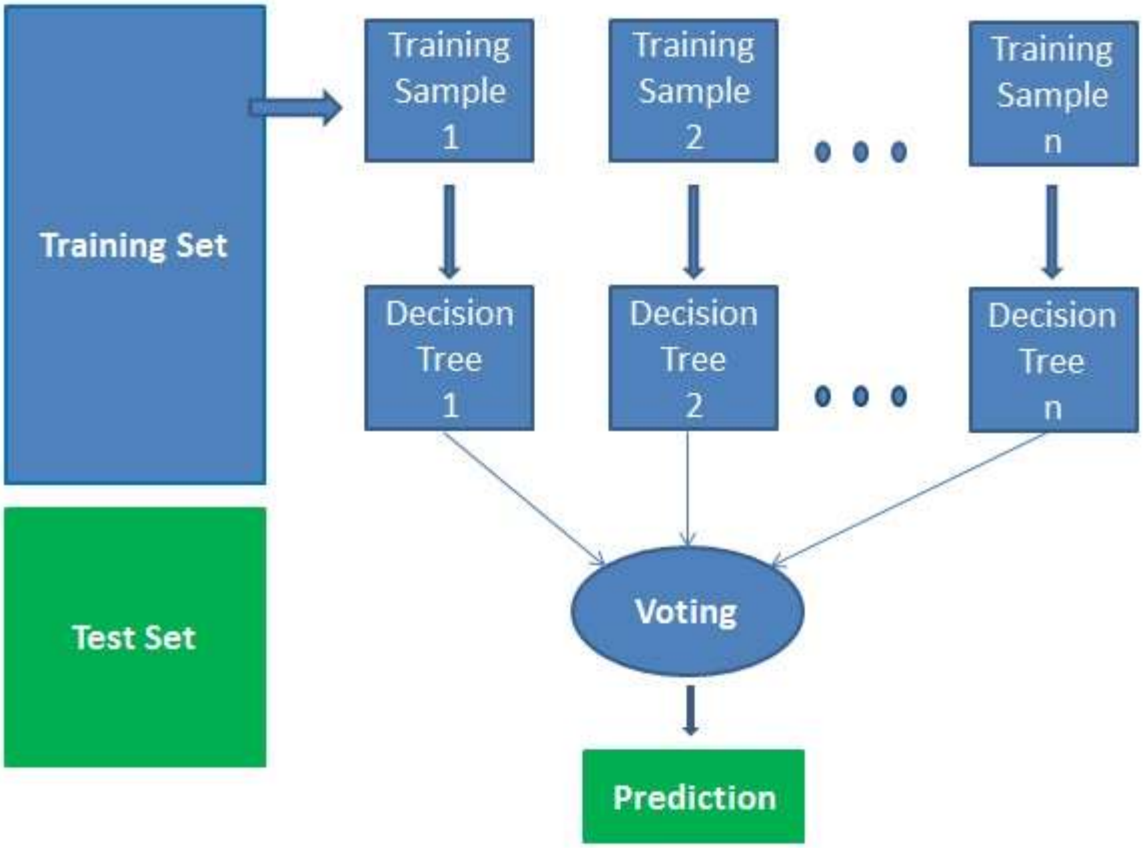
Random Forest Algorithm

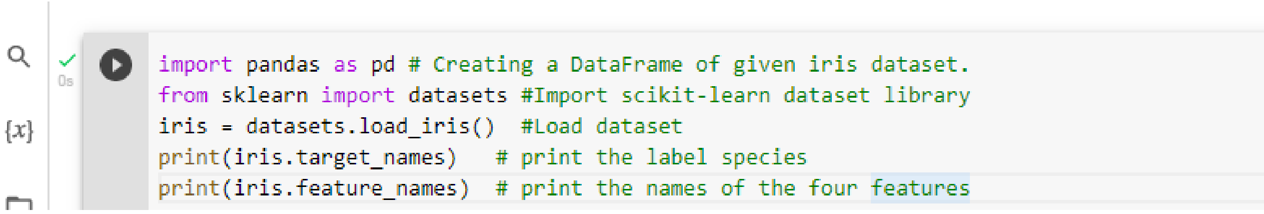
- Random forests is a supervised learning algorithm.
- It can be used both for classification and regression.
- It is also the most flexible and easy to use algorithm.
- A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.
- Random forests has a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases.

How does the Algorithm Work?

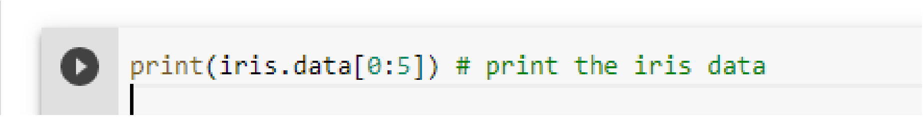
It works in four steps:

1. Select random samples from a given dataset.
2. Construct a decision tree for each sample and get a prediction result from each decision tree.
3. Perform a vote for each predicted result.
4. Select the prediction result with the most votes as the final prediction.

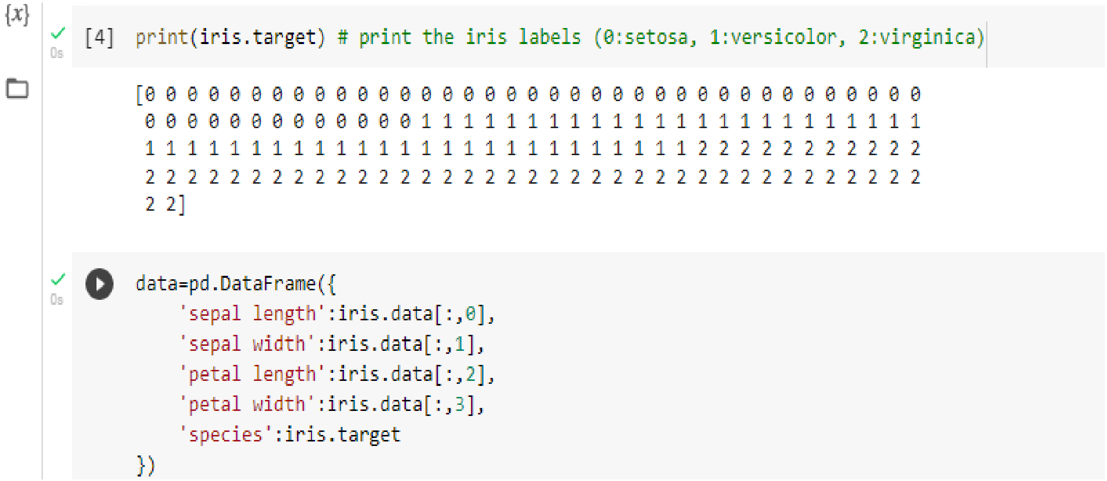




```python
import pandas as pd # Creating a DataFrame of given iris dataset.
from sklearn import datasets #Import scikit-learn dataset library
iris = datasets.load_iris()  #Load dataset
print(iris.target_names)   # print the label species
print(iris.feature_names)  # print the names of the four features
```



```python
print(iris.data[0:5]) # print the iris data
```

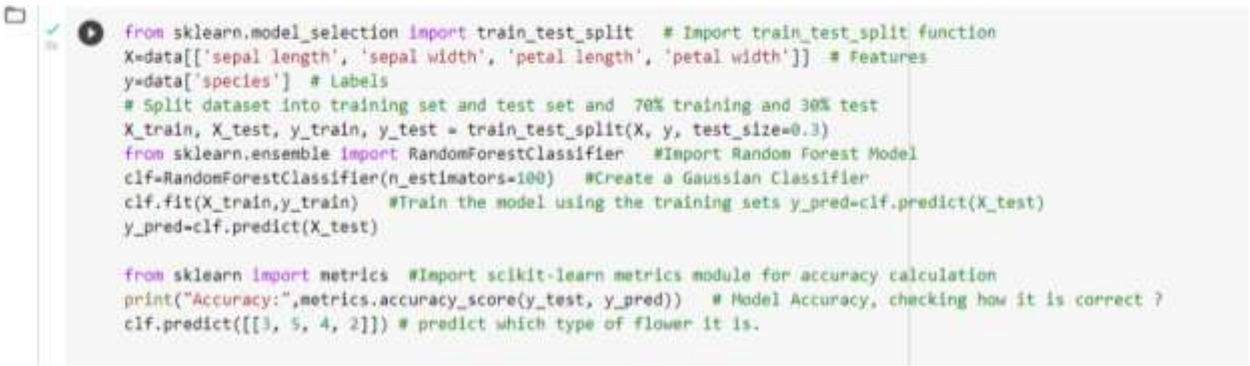


```python
[4] print(iris.target) # print the iris labels (0:setosa, 1:versicolor, 2:virginica)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

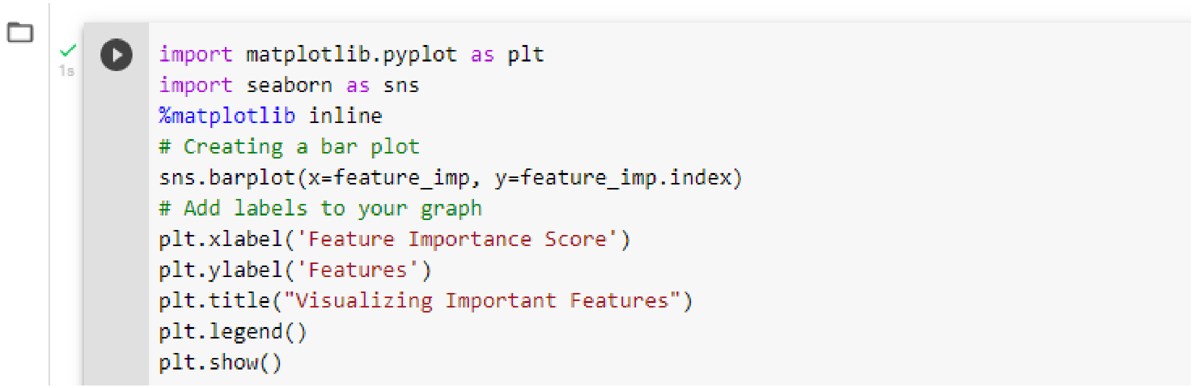
```python
data=pd.DataFrame({
    'sepal length':iris.data[:,0],
    'sepal width':iris.data[:,1],
    'petal length':iris.data[:,2],
    'petal width':iris.data[:,3],
    'species':iris.target
})
```

data.head()

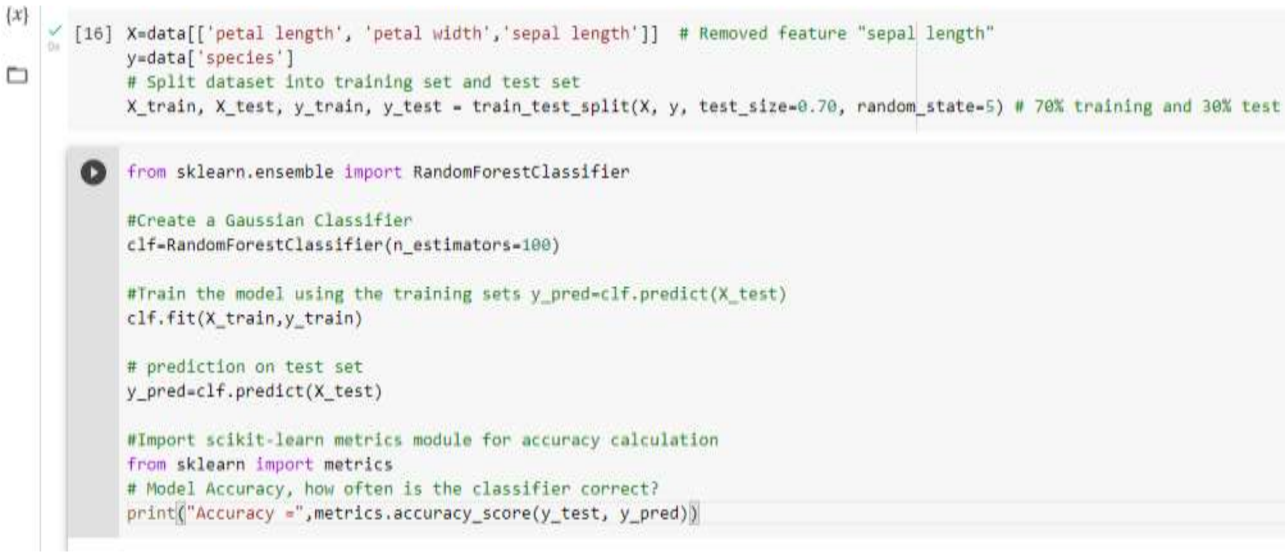


```python
from sklearn.model_selection import train_test_split   # Import train_test_split function
X=data[['sepal length', 'sepal width', 'petal length', 'petal width']]  # Features
y=data['species']  # Labels
# Split dataset into training set and test set and  70% training and 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
from sklearn.ensemble import RandomForestClassifier   #Import Random Forest Model
clf=RandomForestClassifier(n_estimators=100)   #Create a Gaussian Classifier
clf.fit(X_train,y_train)   #Train the model using the training sets y_pred=clf.predict(X_test)
y_pred=clf.predict(X_test)

from sklearn import metrics  #Import scikit-learn metrics module for accuracy calculation
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))   # Model Accuracy, checking how it is correct ?
clf.predict([[3, 5, 4, 2]]) # predict which type of flower it is.
```



```python
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
```



```python
X=data[['petal length', 'petal width','sepal length']]  # Removed feature "sepal length"
y=data['species']
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.70, random_state=5) # 70% training and 30% test
```

```python
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

# prediction on test set
y_pred=clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy =",metrics.accuracy_score(y_test, y_pred))
```